Testing String Superprimitivity in Parallel

Dany Breslauer
Columbia University

CUCS-053-92

Abstract

A string w *covers* another string z if every symbol of z is within some occurrence of w in z. A string is called *superprimitive* if it is covered only by itself, and *quasiperiodic* if it is covered by some shorter string. This paper presents an $O(\log\log n)$ time $\frac{n\log n}{\log\log n}$-processor CRCW-PRAM algorithm that tests if a string is superprimitive. The algorithm is the fastest possible with this number of processors over a general alphabet.

1 Introduction

Quasiperiodicity, as defined by Apostolico and Ehrenfeucht [3], is an avoidable regularity of strings that is strongly related to other regularities such as periods and squares [12]. Apostolico, Farach and Iliopoulos [4] and Breslauer [7] gave linear-time sequential algorithms that tests if a string is superprimitive. Apostolico and Ehrenfeucht [3] presented an algorithm that finds all maximal quasiperiodic substrings of a string.

This paper presents a parallel algorithm that tests if a string of length n is superprimitive in $O(\log\log n)$ time on a $\frac{n\log n}{\log\log n}$-processor CRCW-PRAM. This is the first efficient parallel algorithm for this problem. The algorithm works under the general alphabet assumption where the only access it has to the input string is by pairwise comparisons of symbols.

A parallel algorithm is said to achieve *an optimal speedup* if its time-processor product is the same as the running time of the fastest sequential algorithm for the problem. We show that any $\frac{n\log n}{\log\log n}$-processor parallel superprimitivity testing algorithm over a general alphabet must take at least $\Omega(\log\log n)$ time. Thus, the algorithm presented in this paper is a factor of $\log n$ processors away from optimality. Note that there exists a trivial constant time superprimitivity testing algorithm that uses n^2 processors.

The superprimitivity testing algorithm follows techniques that were used in solving several other parallel string problems [1, 2, 6, 10]. In particular, it uses the parallel string matching algorithm of Breslauer and Galil [8] as a procedure that solves several string matching problems simultaneously and then combines the results of the string matching problems into an answer to the superprimitivity problem.

The paper is organized as follow. Section 2 gives basic definitions and properties of strings. Section 3 overviews the parallel algorithms and tools that are used in the superprimitivity testing algorithm. Section 4 describes the basic step which is used by the superprimitivity testing algorithm in Section 5. Section 6 gives the lower bound.

2 Properties of Strings

A string w *covers* a string z if for every position $i \in \{1, \cdots, |z|\}$ of z there exists an occurrence of w starting at some position j of z such that $1 \leq j \leq i \leq j + |w| - 1 \leq |z|$. A string z is called *superprimitive* if it is covered only by itself, and *quasiperiodic* if it is covered by a string w such that $w \neq z$. A superprimitive string w that covers a string z is called a *quasiperiod* of z.

A string z has a *period* w if z is a prefix of w^k for some integer k. Alternatively, a string w is a period of a string z if $z = w^l v$ and v is a possibly empty prefix of w. The shortest period of a string z is called *the period* of z. Clearly, a string is always a period of itself.

We say that a non-empty string w is a *border* of a string z if z starts and ends with an occurrence of w. That is, $z = uw$ and $z = wv$ for some possibly empty strings u and v. Clearly, a string is always a border of itself. This border is called the trivial border.

We describe next few simple facts which are used in the superprimitivity testing algorithm. Most of these facts were used in the sequential algorithms [4, 7] where their proofs can be found.

Fact 2.1 *A string z has a period of length π, such that $\pi < |z|$, if and only if it has a non-trivial border of length $|z| - \pi$.*

Fact 2.2 *If a string w covers a string z then w is a border of z.*

Note that by the last fact any cover of a string z can be represented by a single integer that is the length the border of z.

Fact 2.3 *If a string w covers a string z and another string v, such that $|w| \leq |v|$, is a border of z then w covers also v.*

Fact 2.4 *If a string z is covered by two strings w and v, such that $|w| \leq |v|$, then w covers v. Therefore, a string cannot have two different quasiperiods w and v.*

Fact 2.5 *If a string z has a border w, such that $2|w| \geq |z|$, then w covers z.*

Proof: The string w covers the first half of the string z since it is a prefix of z and the last half of the string z since it is also a suffix. Therefore, all symbols of z are covered by w. □

3 The CRCW-PRAM Model

The algorithms described in this paper are for the concurrent-read concurrent write parallel random access machine model. We use the weakest version of this model called the *common CRCW-PRAM.* In this model many processors have access to a shared memory. Concurrent read and write operations are allowed at all memory locations. If several processors attempt to write simultaneously to the same memory location, it is assumed they always attempt to write the same value.

The superprimitivity testing algorithm uses the following previously known algorithms:

1. A parallel string matching algorithm that finds all occurrences of a given pattern in a given text. The input to the string matching algorithm consists of two strings: $pattern[1..m]$ and $text[1..n]$ and the output is a Boolean array $match[1..n]$ that has a "*true*" value in each position where an occurrence of the pattern starts in the text. We use the Breslauer and Galil [8] parallel string matching algorithm that takes $O(\log\log m)$ time on a $\frac{n}{\log\log m}$-processor CRCW-PRAM. This algorithm is the fastest optimal parallel string matching algorithm on a general alphabet as implied by a lower bound of Breslauer and Galil [9].

2. The parallel algorithm of Breslauer and Galil [10] that finds all periods of a string of length n in $O(\log\log n)$ time on a $\frac{n}{\log\log m}$-processor CRCW-PRAM. The output of this algorithm is a Boolean array $periods[1..n]$ that has a "*true*" value at each position which is a period of the input string.

3. The algorithm of Fich, Ragde and Wigderson [11] to compute the minimum of n integers between 1 and n in constant time using an n-processor CRCW-PRAM.

One of the major issues in the design of a PRAM algorithms is the assignment of processors to their tasks. We ignore this issue in this paper and use a general theorem that states that the assignment can be done.

Theorem 3.1 *(Brent [5]) Any synchronous parallel algorithm of time t that consists of a total of x elementary operations can be implemented on p processors in $\lceil x/p \rceil + t$ time.*

4 The Basic Step

This section shows how to test efficiently whether a given string w covers another string z.

Theorem 4.1 *Given two string z and w, there exists an algorithm that tests whether w covers z in $O(\log\log|z|)$ time on a $\frac{|z|}{\log\log|z|}$-processor CRCW-PRAM.*

Proof: The algorithm has two steps:

1. Using Breslauer and Galil's [8] string matching algorithm, find all occurrences of w in z. This step takes $O(\log\log|z|)$ time and uses $\frac{|z|}{\log\log|z|}$ processors.

2. Using Fich, Ragde and Wigderson's [11] integer minima algorithm verify that each symbol of z is within an occurrence of w. This step takes constant time and uses $|z|$ processors. It can be done as follows:

   The string $|z|$ is partitioned into consecutive blocks of length $|w|$. The computation proceeds simultaneously in each block.

   The position in z of the first and last occurrences of w in each block are found using the using Fich, Ragde and Wigderson's [11] integer minima algorithm. All symbols of z which are between the first and last occurrences of w in the same block are obviously covered. All that remains to check is whether the symbols between the last occurrence of w in each block and the first occurrence of w in the next block are also covered by testing if the distance between these occurrences is smaller than or equal to $|w|$. A special attention is needed for the first and last blocks where the algorithm checks is an occurrence starts at positions number 1 and $|z| - |w| + 1$ of z. $\square$

5 The Superprimitivity Test

This section describes the parallel superprimitivity test algorithm.

Theorem 5.1 *There exists an algorithm that computes the quasiperiod of a string z in $O(\log\log|z|)$ time on a $\frac{|z|\log|z|}{\log\log|z|}$-processors CRCW-PRAM.*

Proof: The algorithm consists of four steps.

1. Compute all borders of z using Breslauer and Galil's [10] algorithm that finds all periods of a string. Recall that by Fact 2.2 if w covers z then w must be a border of z and by Fact 2.1 there is a one-to-one correspondence between the borders and the periods of a string.

2. Partition the borders of z into intervals $[2^i..2^{i+1}-1]$ according to their length. If there is more than one borders of z whose length is in the same interval then by Fact 2.5 the shorter border covers the longer one. By Fact 2.4 only the shortest border in each interval is a candidate for the quasiperiod of z. The shortest border in each interval can be found in constant time and $|z|$ by using Fich, Ragde and Wigderson's [11] integer minima algorithm in each block simultaneously.

3. In each interval simultaneously, check if the shortest border in the interval covers z.

4. The shortest border that was found to covers z is the quasiperiod of z. $\square$

6 The Lower Bound

We prove a lower bound for testing if a string is superprimitive by a reduction to the lower bound for string matching by Breslauer and Galil [9]. That lower bound is on the number of comparison rounds an algorithm that computes the period of a string has to perform when there are p comparisons in each round. The lower bounds holds for the CRCW-PRAM model in case of a general alphabet where the only access an algorithm has to the input strings is by pairwise comparisons of symbols.

Breslauer and Galil [9] show that an adversary can fool any algorithm which claims to test if a string has a period which is shorter than half of its length in less than $\Omega(\lceil \frac{n}{p} \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ rounds of p comparisons each. Without going into the detail of that lower bound, we use the fact that the adversary of Breslauer and Galil [9] answers the comparisons in each round in such a way that after $\Omega(\lceil \frac{n}{p} \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ rounds it is still possible that the input string has a period that is shorter than half of its length or that is does not have any such period. In the latter case there is at least one symbol of the string that does not appear anywhere else.

Lemma 6.1 *The string generated by Breslauer and Galil's [9] adversary is superprimitive if and only if it does not have a period that is shorter than half of its length.*

Proof: If a string has a period that is shorter than half of its length, then by Fact 2.1 it has a border that is longer than half of its length and by Fact 2.5 is quasiperiodic. On the other hand, if a string has a symbol that appears only once, then it is superprimitive. □

Theorem 6.2 *Any comparison based parallel string superprimitivity test with p comparisons in each round must take at least* $\Omega(\lceil \frac{n}{p} \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ *rounds.*

Proof: By Lemma 6.1 the lower bound of Breslauer and Galil [9] holds also for superprimitivity testing. □

Corollary 6.3 *The algorithm described in Section 5 is the fastest possible with the number of processors used.*

Proof: Substitute $p = \frac{n \log n}{\log\log n}$ in Theorem 6.2. □

7 Acknowledgments

I would like to thank Alberto Apostolico for helpful discussions.

References

[1] A. Apostolico and D. Breslauer. An Optimal $O(\log\log n)$ Time Parallel Algorithm for Detecting all Squares in a String. Technical Report CUCS-040-92, Computer Science Dept., Columbia University, 1992.

[2] A. Apostolico, D. Breslauer, and Z. Galil. Optimal Parallel Algorithms for Periods, Palindromes and Squares. In *Proc. 19th International Colloquium on Automata, Languages, and Programming*. Springer-Verlag, Berlin, Germany, 1992. 296–307.

[3] A. Apostolico and A. Ehrenfeucht. Efficient Detection of Quasiperiodicities in Strings. Technical Report 90.5, The Leonadro Fibonacci Institute, Trento, Italy, 1990.

[4] A. Apostolico, M. Farach, and C. S. Iliopoulos. Optimal superprimitivity testing for strings. *Inform. Process. Lett.*, 39:17–20, 1991.

[5] R. P. Brent. Evaluation of general arithmetic expressions. *J. Assoc. Comput. Mach.*, 21:201–206, 1974.

[6] D. Breslauer. *Efficient String Algorithmics.* PhD thesis, Dept. of Computer Science, Columbia University, New York, NY, 1992.

[7] D. Breslauer. An On-Line String Superprimitivity Test. *Inform. Process. Lett.*, to appear.

[8] D. Breslauer and Z. Galil. An optimal $O(\log\log n)$ time parallel string matching algorithm. *SIAM J. Comput.*, 19(6):1051–1058, 1990.

[9] D. Breslauer and Z. Galil. A Lower Bound for Parallel String Matching. *SIAM J. Comput.*, 21(5):856–862, 1992.

[10] D. Breslauer and Z. Galil. Finding all periods and initial palindromes of a string in parallel. Technical Report CUCS-017-92, Computer Science Dept., Columbia University, 1992.

[11] F. E. Fich, R. L. Ragde, and A. Wigderson. Relations between concurrent-write models of parallel computation. In *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, pages 179–189, 1984.

[12] M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA., U.S.A., 1983.